Loops (While and For)

CSE 1310 – Introduction to Computers and Programming
Vassilis Athitsos
University of Texas at Arlington
Motivation for Loops - First Example

• We have written a program for calculating the area and circumference of a circle.
  – Problem: we need to re-run the program every time we want to compute values for a new radius.
  – The user should be able to keep entering values, as long as they want.

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        System.out.printf("Please enter the radius: ");
        double radius = in.nextDouble();
        double circumference = 2 * Math.PI * radius;
        double area = Math.PI * Math.pow(radius, 2);
        System.out.printf("The circumference is %.2f.\n", circumference);
        System.out.printf("The area is %.2f.\n", area);
    }
}
```
Suppose we want to write programs that ask the user to input an integer \( N \) and then do one (or more) of the following:

- Print out all integers between 0 and \( N \).
- Figure out if \( N \) is a prime number.
- Print out all prime numbers between 1 and \( N \).
- ...

The elements of Java that we have covered so far are not sufficient for writing such programs.

What is missing: the ability to repeat some instructions as many times as we want.
Example of a **while** loop: Printing Numbers from 1 to N

- Write a program that:
  - Asks the user to enter an integer N.
  - Prints all integers from 1 to N.
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        System.out.printf("Please enter an integer: ");
        int N = in.nextInt();

        int i = 1;
        while (i <= N) {
            System.out.printf("%d
", i);
            i = i+1;
        }

        System.out.printf("done with the while loop.
");
    }
}
while loops

• A `while` loop is defined as follows:

```c
while (condition)
{
    line 1
    line 2
    ...
    line n
}
```

• `condition` is a **boolean expression** (that can be equal to **true** or **false**).

• Line 1, line 2, ..., line n are called the **body** of the `while` loop.
while loops

- A **while** loop is defined as follows:

```java
while (condition)
{
    line 1
    line 2
    ...
    line n
}
```

- Meaning: as long as **condition** is true, keep executing the body of the loop (lines 1, ..., n).
while loop execution

```plaintext
while (condition) {
    line 1
    line 2
    ...
    line n
}
first line after loop
```

• This is how a while loop gets executed:
  – Step 1: evaluate `condition`.
  – Step 2: If `condition` is false, go to the first line after the loop.
  – Step 3: If `condition` is true, execute the body of the while loop, and go back to step 1.
Example of a **while** loop:

Printing Numbers from 1 to N

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        System.out.printf("Please enter an integer: ");
        int N = in.nextInt();

        int i = 1;
        while (i <= N) {
            System.out.printf("%d\n", i);
            i = i+1;
        }

        System.out.printf("done with the while loop.\n");
    }
}
```
Example of a **while** loop:

**Printing Numbers from 1 to N**

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        System.out.printf("Please enter an integer: ");
        int N = in.nextInt();

        int i = 1;
        while (i <= N) {
            System.out.printf("%d
", i);
            i = i+1;
        }

        System.out.printf("done with the while loop.
");
    }
}
```

What is the **condition** for this while loop?
Example of a **while** loop:
Printing Numbers from 1 to N

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        System.out.printf("Please enter an integer: ");
        int N = in.nextInt();

        int i = 1;
        while (i <= N) {
            System.out.printf("%d\n", i);
            i = i+1;
        }

        System.out.printf("done with the while loop.\n");
    }
}
```

What is the **condition** for this while loop?

i <= N
Example of a **while** loop: Printing Numbers from 1 to N

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        System.out.printf("Please enter an integer: ");
        int N = in.nextInt();

        int i = 1;
        while (i <= N) {
            System.out.printf("%d\n", i);
            i = i+1;
        }

        System.out.printf("done with the while loop.\n");
    }
}
```

What is the **body** of this while loop?
Example of a while loop:
Printing Numbers from 1 to N

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        System.out.printf("Please enter an integer: ");
        int N = in.nextInt();

        int i = 1;
        while (i <= N)
        {
            System.out.printf("%d\n", i);
            i = i+1;
        }

        System.out.printf("done with the while loop.\n");
    }
}
```

What is the body of this while loop?
The lines shown in red on this slide. (Everything between the curly braces under the while line).
Example of a **while** loop:

Printing Numbers from 1 to N

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        System.out.printf("Please enter an integer: ");
        int N = in.nextInt();

        int i = 1;
        while (i <= N) {
            System.out.printf("%d
", i);
            i = i+1;
        }

        System.out.printf("done with the while loop.");
    }
}
```

Please enter an integer: 5
1
2
3
4
5
done with the while loop.
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        System.out.printf("Please enter an integer: ");
        int N = in.nextInt();

        int i = 1;
        while (i <= N)
        {
            System.out.printf("%d\n", i);
        }

        System.out.printf("done with the while loop.\n");
    }
}
Common Bug: Infinite Loop

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        System.out.printf("Please enter an integer: ");
        int N = in.nextInt();

        int i = 1;
        while (i <= N) {
            System.out.printf("%d
", i);
        }

        System.out.printf("done with the while loop.\n");
    }
}
```

To quit an infinite loop in NetBeans:

Select Run->Stop Build/Run

If you do not do that, you will not be able to run your (corrected) program again.
Designing a `while` loop

• When you design a `while` loop, you need to make sure that the loop will terminate exactly when needed, not before, and not after.

• You will need to define a boolean condition, that determines exactly when to stay in the loop and when to exit.

• You need to update variables within the body of the loop, as needed, to make sure that the boolean condition will evaluate to the right thing.
Example of a **while** loop:
Determining if an Integer N is Prime

- Write a program that:
  - Asks the user to enter an integer N.
  - Prints whether N is prime.
Example of a **while** loop:

Determining if an Integer N is Prime

• Write a program that:
  – Asks the user to enter an integer N.
  – Prints whether N is prime.

• Strategy for determining if N is prime:
Example of a `while` loop:
Determining if an Integer \( N \) is Prime

• Write a program that:
  – Asks the user to enter an integer \( N \).
  – Prints whether \( N \) is prime.

• Strategy for determining if \( N \) is prime:
  – For every number \( K \) between 2 and \( N-1 \), check if \( K \) divides \( N \).
Example of a **while** loop: Determining if an Integer N is Prime

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Enter an integer N: ");
        int N = in.nextInt();

        if (N is prime) {
            System.out.printf("%d is prime.\n", N);
        } else {
            System.out.printf("%d is not prime.\n", N);
        }
    }
}
```
Example of a **while** loop:
Determining if an Integer N is Prime

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Enter an integer N: ");
        int N = in.nextInt();

        if (N is prime) {
            System.out.printf("%d is prime.\n", N);
        } else {
            System.out.printf("%d is not prime.\n", N);
        }
    }
}
```

Are we done?

No, because the code does not have anything for figuring out if N is prime.

However, writing code like this is a very useful strategy:

- Start with parts of the code that need to be there for sure.
- Then, start adding pieces that are missing.
Example of a **while** loop:
Determining if an Integer N is Prime

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Enter an integer N: ");
        int N = in.nextInt();

        if (N is prime)
        {
            System.out.printf("%d is prime.\n", N);
        }
        else
        {
            System.out.printf("%d is not prime.\n", N);
        }
    }
}

**VERY IMPORTANT TIP:**
(you should be doing it throughout this course):

If there is a place in your code where you need some information that you don't have:

- Create a variable.
- Write code so that this variable has the information you need, at the point where you need it.
Example of a **while** loop:

Determining if an Integer N is Prime

import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Enter an integer N: ");
        int N = in.nextInt();

        if (N is prime) {
            System.out.printf("%d is prime.\n", N);
        } else {
            System.out.printf("%d is not prime.\n", N);
        }
    }
}
Example of a **while** loop:  
Determining if an Integer N is Prime

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Enter an integer N: ");
        int N = in.nextInt();

        ?? N_is_prime;

        if (N_is_prime)
        {
            System.out.printf("%d is prime.\n", N);
        }
        else
        {
            System.out.printf("%d is not prime.\n", N);
        }
    }
}
```

Here, the information we need is:  
is N prime?

So, we need to create a variable.

Let's call it  
N_is_prime

What is the type?
**Example of a while loop:**

**Determining if an Integer N is Prime**

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Enter an integer N: ");
        int N = in.nextInt();

        N_is_prime;

        if (N is prime)
        {
            System.out.printf("%d is prime.\n", N);
        }
        else
        {
            System.out.printf("%d is not prime.\n", N);
        }
    }
}
```

**ANOTHER IMPORTANT TIP:**

(you should be doing it throughout this course):

To figure out what type a variable should be:

Think about all possible values that this variable should be able to take, to handle all cases that you care about.
Example of a \textbf{while} loop:
Determining if an Integer N is Prime

\begin{verbatim}
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Enter an integer N: ");
        int N = in.nextInt();

        boolean N_is_prime;

        if (N is prime)
            System.out.printf("%d is prime.\n", N);
        else
            System.out.printf("%d is not prime.\n", N);
    }
}
\end{verbatim}

N\_is\_prime can take values \texttt{true} or \texttt{false}.

Therefore, \texttt{N\_is\_prime} should be of type \texttt{boolean}. 
Example of a **while** loop:

Determining if an Integer N is Prime

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Enter an integer N: ");
        int N = in.nextInt();

        boolean N_is_prime;
        // CODE NEEDED HERE.
        if (N is prime)
        {
            System.out.printf("%d is prime.\n", N);
        }
        else
        {
            System.out.printf("%d is not prime.\n", N);
        }
    }
}
```

N_is_prime can take values **true** or **false**.

Therefore, N_is_prime should be of type **boolean**.

Next: writing code to make sure N_is_prime has the right value where we need it.
import java.util.Scanner;
public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Enter an integer N: ");
        int N = in.nextInt();
        boolean N_is_prime = true;
        int i = 2;
        while (i < N) {
            if (N % i == 0) {
                N_is_prime = false;
            }
            i++;
        }
        if (N_is_prime) {
            System.out.printf("%d is prime. \n", N);
        } else {
            System.out.printf("%d is not prime. \n", N);
        }
    }
}
When a boolean value depends on a smoking gun:

– Initialize the boolean variable to the value it should get if we find no smoking gun.

– Do a loop, where you test all possible smoking guns. If you find a smoking gun, flip the value of the boolean variable.

Mishandling smoking gun problems is (unfortunately) a very common mistake in this course.

```java
import java.util.Scanner;
public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Enter an integer N: ");
        int N = in.nextInt();
        boolean N_is_prime = true;
        int i = 2;
        while (i < N) {
            if (N % i == 0) {
                N_is_prime = false;
            }
            i++;
        }
        if (N_is_prime) {
            System.out.printf("%d is prime.\n", N);
        } else {
            System.out.printf("%d is not prime.\n", N);
        }
    }
}```
A classic mistake in smoking gun problems:

- Setting the Boolean variable at every iteration.

Why is this a mistake?
• A classic mistake in smoking gun problems:
  – Setting the Boolean variable at every iteration.

• Effectively, this makes the entire loop useless.
  – At the end, the Boolean variable will be set at the last iteration.
  – The calculations of all previous iterations are ignored.

• The code on the left is **incorrect**, illustrates this classic mistake.
  – N_is_prime is set to true every time N % i != 0.
Another classic mistake in smoking gun problems:
- Declaring the Boolean variable within the body of the loop.

Why is this a mistake?

// This code is incorrect
import java.util.Scanner;
public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Enter an integer N: ");
        int N = in.nextInt();
        int i = 2;
        while (i < N)
        {
            boolean N_is_prime = true;
            if (N % i == 0)
            {
                N_is_prime = false;
            }
            i++;
        }
        if (N_is_prime)
        {
            System.out.printf("%d is prime.\n", N);
        }
        else
        {
            System.out.printf("%d not prime.\n", N);
        }
    } // This code is incorrect
Another classic mistake in smoking gun problems:
  – Declaring the Boolean variable within the body of the loop.

If you make that mistake, Java will give you an error here:
  – If your variable has been declared inside the loop, then it is not defined outside the loop.

The code on the left is incorrect, illustrates this classic mistake.
Example of a \texttt{for} loop: Printing Numbers from 1 to N

- Write a program that:
  - Asks the user to enter an integer N.
  - Prints all integers from 1 to N.
Example of a **for** loop: Printing Numbers from 1 to N

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        System.out.printf("Please enter an integer: ");
        int N = in.nextInt();

        for (int i = 1; i <= N; i++)
        {
            System.out.printf("%d\n", i);
        }

        System.out.printf("done with the for loop.\n");
    }
}
```
for loops

• A for loop can be defined as follows (note: this definition will be extended when we talk about lists).

```java
for (int var = init_value; condition; update)
{
    line 1
    line 2
    ...
    line n
}
```

• Line 1, line 2, ..., line n are called the body of the for loop.
for loops

• A for loop can be defined as follows (note: this definition will be extended when we talk about lists).

```c
for (int var = init_value; condition; update)
{
    line 1
    line 2
    ...
    line n
}
```

• The condition is a boolean expression, that typically compares `var` to some value.

• E.g.: `var <= N`. 
for loops

• A for loop can be defined as follows (note: this definition will be extended when we talk about lists).

```c
for (int var = init_value; condition; update)
{
    line 1
    line 2
    ...
    line n
}
```

• The update typically changes the value of `var`.
• Most common case: `var++`.
• Another example: `var = var - 3`
for loop execution

```cpp
for (int var = init_value; condition; update) {
    line 1
    line 2
    ...
    line n
}
```

first line after loop

• This is how a for loop gets executed:
  – Step 1: `var = init_value;`
  – Step 2: If `condition` is false, go to first line after the loop.
  – Step 3: execute the body of the loop (lines 1 to n).
  – Step 4: execute the update, and go to step 2.
Example of a **for** loop:

Printing Numbers from 1 to N

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        System.out.printf("Please enter an integer: ");
        int N = in.nextInt();

        for (int i = 1; i <= N; i++)
        {
            System.out.printf("%d\n", i);
        }

        System.out.printf("done with the for loop.\n");
    }
}
```

What is the **condition** for this for loop?

What is the **update** for this for loop?

What is the **body** for this for loop?
Example of a *for* loop: Printing Numbers from 1 to N

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        System.out.printf("Please enter an integer: ");
        int N = in.nextInt();

        for (int i = 1; i <= N; i++) {
            System.out.printf("%d
", i);
        }

        System.out.printf("done with the for loop.\n");
    }
}
```

What is the **condition** for this for loop?

i <= N

What is the **update** for this for loop?

i++

What is the **body** for this for loop?

The *printf* line.
import java.util.Scanner;
public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int N = in.nextInt();

        for (int i = 0; i <= N; i += 13) {
            System.out.printf("%d\n", i);
        }
        System.out.printf("printed all numbers between 0 and %d\n", N);
        System.out.printf("that are divisible by 13.\n", N);
    }
}
import java.util.Scanner;
public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int N = in.nextInt();

        for (int i = 0; i <= N; i += 13)
        {
            System.out.printf("%d\n", i);
        }
        System.out.printf("printed all numbers between 0 and %d\n", N);
        System.out.printf("that are divisible by 13.\n", N);
    }
}

Please enter an integer: 30
0
13
26
printed all numbers between 0 and 30 that are divisible by 13.
import java.util.Scanner;
public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int N = in.nextInt();

        for (int i = N; i >= 0; i -= 2) {
            System.out.printf("%d
", i);
        }
        System.out.printf("Counting down %d to 0, with step 2.
", N);
    }
}

Please enter an integer: 5

Example output:
import java.util.Scanner;
public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int N = in.nextInt();

        for (int i = N; i >= 0; i -= 2) {
            System.out.printf("%d\n", i);
        }
        System.out.printf("Counting down %d to 0, with step 2.\n", N);
    }
}

Please enter an integer: 5
5
3
1
Counting down 5 to 0, with step 2.
• Write a program that:
  – Asks the user to enter a word.
  – Prints each letter of that word on a separate line.
for Loop With a String: Example 1

import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter a word: ");
        String word = in.next();

        for (int i = 0; i < word.length(); i++)
        {
            System.out.printf("%s\n", word.charAt(i));
        }
    }
}

Please enter a word: hello

Example output:
for Loop With a String: Example 1

```
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter a word: ");
        String word = in.next();

        for (int i = 0; i < word.length(); i++)
        {
            System.out.printf("%s
", word.charAt(i));
        }
    }
}
```

Example output:
Please enter a word: hello
h
e
l
l
o
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter a word: ");
        String word = in.next();

        int i = 0;
        while (i < word.length())
        {
            System.out.printf("%s\n", word.charAt(i));
            i++;
        }
    }
}

Example output:
Please enter a word: hello
h
 e
l
 l
 o
for Loop With a String: Example 2

• Write a program that:
  – Asks the user to enter a word.
  – Starting from the first letter, it prints every other letter of the word. The letters should be printed on the same line, not one per line.
  – For example, for "Sunday" it should print "Sna".
for Loop With a String: Example 2

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter a word: ");
        String word = in.next();

        for (int i = 0; i < word.length(); i+=2)
        {
            System.out.printf("%s\n", word.charAt(i));
        }
    }
}
```

Example output:

Please enter a word: Sunday
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter a word: ");
        String word = in.next();

        for (int i = 0; i < word.length(); i+=2)
        {
            System.out.printf("%s\n", word.charAt(i));
        }
    }
}

Not what we want. We want all letters on the same line, like "Sna".

Example output:
Please enter a word: Sunday
S
n
a
for Loop With a String: Example 2

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter a word: ");
        String word = in.next();

        for (int i = 0; i < word.length(); i+=2) {
            System.out.printf("%s", word.charAt(i));
        }
        System.out.printf("\n");
    }
}
```

If we remove `\n` from the `printf`, it works. We just need to print a new line at the end of the program.

Example output: Please enter a word: Sunday
Sna
```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter a word: ");
        String word = in.next();

        int i = 0;
        while (i < word.length()) {
            System.out.printf("%s", word.charAt(i));
            i += 2;
        }
        System.out.printf("\n");
    }
}
```

Example output:
```
Please enter a word: Sunday
Sna
```
for Loop With a String: Example 3

• Write a program that:
  – Asks the user to enter a word.
  – Prints the letters of the string backwards. The letters should be printed on the same line, not one per line.
  – For example, for "Sunday" it should print "yadnuS".
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter a word: ");
        String word = in.next();

        for (int i = word.length() - 1; i >= 0; i--)
            System.out.printf("%s", word.charAt(i));
        System.out.printf("\n");
    }
}

(Very) important things:
• initial value of i.
• terminating condition.
• update.

Example output: Please enter a word: Sunday yadnuS
while Loop Version

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter a word: ");
        String word = in.next();

        int i = word.length() - 1;
        while (i >= 0)
        {
            System.out.printf("%s", word.charAt(i));
            i--;
        }
        System.out.printf("\n");
    }
}
```

Example output: Please enter a word: Sunday yadnuS
for Loop With a String: Example 4

• Write a program that:
  – Asks the user to enter a word.
  – Counts the number of times the letter 'a' appears in the word.
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter a word: ");
        String word = in.next();

        int counter = 0;
        for (int i = 0; i < word.length(); i++)
        {
            char c = word.charAt(i);
            if (c == 'a')
            {
                counter++;
            }
        }

        System.out.printf("The letter a occurs %d times.\n", counter);
    }
}

Please enter a word: January
The letter a occurs 2 times.
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter a word: ");
        String word = in.next();

        int counter = 0;
        for (int i = 0; i < word.length(); i++) {
            char c = word.charAt(i);
            if (c == 'a') {
                counter++;
            }
        }
        System.out.printf("The letter a occurs %d times.\n", counter);
    }
}

• This is the classic **counter** problem (we will see MANY such problems).
  – We must count how many times something happens.
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter a word: ");
        String word = in.next();

        int counter = 0;
        for (int i = 0; i < word.length(); i++)
        {
            char c = word.charAt(i);
            if (c == 'a')
            {
                counter++;
            }
        }
        System.out.printf("The letter a occurs %d times.\n", counter);
    }
}

• To solve the counter problem:
  – Initialize a counter variable to 0, before the loop.
  – Do a loop, where you increment the counter every time you find what you are looking for.
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter a word: ");
        String word = in.next();

        int counter = 0;
        for (int i = 0; i < word.length(); i++) {
            char c = word.charAt(i);
            if (c == 'a') {
                counter++;
            }
        }
        System.out.printf("The letter a occurs %d times.\n", counter);
    }
}
The **break** statement

- The **break** statement forces termination of the current **while** loop or **for** loop.
- Example: print the first number >= N that is divisible by 13.

```java
import java.util.Scanner;
public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int N = in.nextInt();

        int i = N;
        while(true) {
            if (i % 13 == 0) {
                System.out.printf("%d is the first integer >= %d that is divisible by 13.\n", i, N);
                break;
            }
            i++;
        }
    }
}
```
Example output:

Please enter an integer: 62
65 is the first integer >= 62 that is divisible by 13.
break

while (condition) {
    line 1
    line 2
    ...
    line n
}
first line after loop

• Suppose that we execute a **break** within the body of the while loop.

• What line of code will be executed next?
break

while (condition) {
    line 1
    line 2
    ...
    line n
}
first line after loop

• Suppose that we execute a **break** within the body of the while loop.

• What line of code will be executed next?
  – The first line after the loop.
for (int var = init_value; condition; update)
{
    line 1
    line 2
    ...
    line n
}
first line after loop

• Suppose that we execute a **break** within the body of the for loop.
• What line of code will be executed next?
break

```java
for (int var = init_value; condition; update) {
    line 1
    line 2
    ...
    line n
}
```

first line after loop

• Suppose that we execute a `break` within the body of the for loop.

• What line of code will be executed next?
  – The first line after the loop.
for (int var = init_value; condition; update) {
    line 1
    line 2
    ...
    line n
}
first line after loop

• Suppose that we execute a **break** within the body of the for loop.
• What if there is no first line after the loop?
break

for (int var = init_value; condition; update) {
   line 1
   line 2
   ...
   line n
}
first line after loop

• Suppose that we execute a break within the body of the for loop.

• What if there is no first line after the loop?
  – The program will just terminate.
The **continue** statement

- The **continue** statement skips the rest of the body of the loop and goes directly to the next iteration (or to termination).
- Example: print numbers between 1 and N that are divisible by 13.

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int N = in.nextInt();

        for (int i = 1; i <= N; i++)
        {
            if (i % 13 != 0)
            {
                continue;
            }
            System.out.printf("%d
", i);
        }
    }
}
```

Example output:

```
Please enter an integer: 50
```
The `continue` statement

- The `continue` statement skips the rest of the body of the loop and goes directly to the next iteration (or to termination).
- Example: print numbers between 1 and N that are divisible by 13.

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Please enter an integer: ");
        int N = in.nextInt();

        for (int i = 1; i <= N; i++) {
            if (i % 13 != 0) {
                continue;
            }
            System.out.printf("%d
", i);
        }
    }
}
```

Example output:
```
Please enter an integer: 50
13
26
39
```
continue

while (condition) {
    line 1
    line 2
    ...
    line n
}
first line after loop

• Suppose that we execute a `continue` within the body of the while loop.
• What line of code will be executed next?
continue

while (condition)
{
    line 1
    line 2
    ...  
    line n
}

first line after loop

• Suppose that we execute a `continue` within the body of the while loop.
• What line of code will be executed next?
  – `condition`
for (int var = init_value; condition; update) {
    line 1
    line 2
    ...
    line n
}

first line after loop

• Suppose that we execute a continue within the body of the for loop.
• What will happen next?
continue

for (int var = init_value; condition; update)
{
    line 1
    line 2
    ...
    line n
}
first line after loop

• Suppose that we execute a `continue` within the body of the for loop.

• What will happen next?
  – Execute the update.
  – Check the condition, and loop again or exit the loop.
Nested Loops

• A loop can be part of another loop. Such a loop is called a **nested loop**.

• Example 1: Print out the 10x10 multiplication table.
Nested Loops

• A loop can be part of another loop. Such a loop is called a nested loop.
• Example 1: Print out the 10x10 multiplication table.

```java
public class example1 {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++)
            { 
                for (int j = 1; j <= 10; j++)
                    {
                        System.out.printf("%d ", i*j);
                    }
                System.out.printf("\n");
            }
    }
}
```
Nested Loops

- A loop can be part of another loop. Such a loop is called a **nested loop**.
- Example 1: Print out the 10x10 multiplication table.

Output, version 1:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>16</td>
<td>18</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>12</td>
<td>15</td>
<td>18</td>
<td>21</td>
<td>24</td>
<td>27</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td>24</td>
<td>28</td>
<td>32</td>
<td>36</td>
<td>40</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td>30</td>
<td>35</td>
<td>40</td>
<td>45</td>
<td>50</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>12</td>
<td>18</td>
<td>24</td>
<td>30</td>
<td>36</td>
<td>42</td>
<td>48</td>
<td>54</td>
<td>60</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>14</td>
<td>21</td>
<td>28</td>
<td>35</td>
<td>42</td>
<td>49</td>
<td>56</td>
<td>63</td>
<td>70</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>16</td>
<td>24</td>
<td>32</td>
<td>40</td>
<td>48</td>
<td>56</td>
<td>64</td>
<td>72</td>
<td>80</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>18</td>
<td>27</td>
<td>36</td>
<td>45</td>
<td>54</td>
<td>63</td>
<td>72</td>
<td>81</td>
<td>90</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
<td>90</td>
<td>100</td>
</tr>
</tbody>
</table>

Any problem?
### Nested Loops

- A loop can be part of another loop. Such a loop is called a **nested loop**.
- Example 1: Print out the 10x10 multiplication table.

<table>
<thead>
<tr>
<th>Output, version 1:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9 10</td>
</tr>
<tr>
<td>2 4 6 8 10 12 14 16 18 20</td>
</tr>
<tr>
<td>3 6 9 12 15 18 21 24 27 30</td>
</tr>
<tr>
<td>4 8 12 16 20 24 28 32 36 40</td>
</tr>
<tr>
<td>5 10 15 20 25 30 35 40 45 50</td>
</tr>
<tr>
<td>6 12 18 24 30 36 42 48 54 60</td>
</tr>
<tr>
<td>7 14 21 28 35 42 49 56 63 70</td>
</tr>
<tr>
<td>8 16 24 32 40 48 56 64 72 80</td>
</tr>
<tr>
<td>9 18 27 36 45 54 63 72 81 90</td>
</tr>
<tr>
<td>10 20 30 40 50 60 70 80 90 100</td>
</tr>
</tbody>
</table>

Any problem? **The output is correct, but the numbers are not aligned nicely.**
Nested Loops

• A loop can be part of another loop. Such a loop is called a nested loop.
• Example 1: Print out the 10x10 multiplication table.

```
public class example1 {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
            for (int j = 1; j <= 10; j++) {
                System.out.printf("%3d ", i*j);
            }
            System.out.printf("\n");
        }
    }
}
```

Code, version 2:
Nested Loops

• A loop can be part of another loop. Such a loop is called a nested loop.

• Example 1: Print out the 10x10 multiplication table.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>16</td>
<td>18</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>12</td>
<td>15</td>
<td>18</td>
<td>21</td>
<td>24</td>
<td>27</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td>24</td>
<td>28</td>
<td>32</td>
<td>36</td>
<td>40</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td>30</td>
<td>35</td>
<td>40</td>
<td>45</td>
<td>50</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>12</td>
<td>18</td>
<td>24</td>
<td>30</td>
<td>36</td>
<td>42</td>
<td>48</td>
<td>54</td>
<td>60</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>14</td>
<td>21</td>
<td>28</td>
<td>35</td>
<td>42</td>
<td>49</td>
<td>56</td>
<td>63</td>
<td>70</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>16</td>
<td>24</td>
<td>32</td>
<td>40</td>
<td>48</td>
<td>56</td>
<td>64</td>
<td>72</td>
<td>80</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>18</td>
<td>27</td>
<td>36</td>
<td>45</td>
<td>54</td>
<td>63</td>
<td>72</td>
<td>81</td>
<td>90</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
<td>90</td>
<td>100</td>
</tr>
</tbody>
</table>

Output, version 2:
Break and Continue in Nested Loops

for (int var1 = init1; condition1; update1) // start of loop1
{
    ...
    for (int var2 = init2; condition2; update2) // start of loop2
    {
        ...
        ...
    }
} first line after loop2
    ...
}
} first line after loop1

• Suppose some break line belongs to multiple loops.
• If that break line is executed, what line of code do we go to?
Break and Continue in Nested Loops

```java
for (int var1 = init1; condition1; update1) // start of loop1
{
    ...
    for (int var2 = init2; condition2; update2) // start of loop2
    {
        ...
        ...
    }
    first line after loop2
    ...
}
first line after loop1
```

- Suppose some `break` line belongs to multiple loops.
- If that `break` line is executed, what line of code do we go to?
  - The first line after the **innermost loop** containing the `break`. 
• What line is executed after the `break` in this example?
Break and Continue in Nested Loops

```java
for (int var1 = init1; condition1; update1) // start of loop1
{
    ...
    for (int var2 = init2; condition2; update2) // start of loop2
    {
        ...
        break;
        ...
    }
    first line after loop2
    ...
}
first line after loop1
```

• The innermost loop that the `break` belongs to is loop 2.
• The next line is the first line after loop 2 (shown in green).
Break and Continue in Nested Loops

for (int var1 = init1; condition1; update1) // start of loop1
{
    ...
    break;
    ...
    for (int var2 = init2; condition2; update2) // start of loop2
    {
        ...
    }
    first line after loop2
    ...
}
first line after loop1

• What line is executed after the `break` in this example?
Break and Continue in Nested Loops

for (int var1 = init1; condition1; update1) // start of loop1
{
    ...
    break;
    ...
    for (int var2 = init2; condition2; update2) // start of loop2
    {
        ...
    }
    first line after loop2
    ...
}
first line after loop1

• The innermost loop that the `break` belongs to is loop 1.
• The next line is the first line after loop 1 (shown in green).
Break and Continue in Nested Loops

for (int var1 = init1; condition1; update1) // start of loop1
{
    ...
    for (int var2 = init2; condition2; update2) // start of loop2
    {
        ...
        ...
    }
    first line after loop2
    ...
}
first line after loop1

• Suppose some continue line belongs to multiple loops.
• If that continue line is executed, what line of code do we go to?
Break and Continue in Nested Loops

for (int var1 = init1; condition1; update1) // start of loop1
{
    ...
    for (int var2 = init2; condition2; update2) // start of loop2
    {
        ...
        ...
    }
    first line after loop2
    ...
}
first line after loop1

• Suppose some continue line belongs to multiple loops.
• If that continue line is executed, what line of code do we go to?
  – The first line of the innermost loop containing the continue.
Break and Continue in Nested Loops

for (int var1 = init1; condition1; update1) // start of loop1
{
    ...
    for (int var2 = init2; condition2; update2) // start of loop2
    {
        ...
        continue;
        ...
    }
    first line after loop2
    ...
}
first line after loop1

• What happens after continue in this example?
for (int var1 = init1; condition1; update1) // start of loop1
{
    ...
    for (int var2 = init2; condition2; update2) // start of loop2
    {
        ...
        continue;
        ...
    }
    first line after loop2
    ...
}
first line after loop1

• The innermost loop that continue belongs to is loop 2.
• After continue, Java executes update2 and condition2.
Break and Continue in Nested Loops

for (int var1 = init1; condition1; update1) // start of loop1
{
    ...
    for (int var2 = init2; condition2; update2) // start of loop2
    {
        ...
    }
    first line after loop2
    ...
    continue;
    ...
}
first line after loop1

• What happens after continue in this example?
Break and Continue in Nested Loops

for (int var1 = init1; condition1; update1) // start of loop1
{
    ...
    for (int var2 = init2; condition2; update2) // start of loop2
    {
        ...
    }
    first line after loop2
    ...
    continue;
    ...
}
first line after loop1

• The innermost loop that **continue** belongs to is loop 1.
• After **continue**, Java executes **update1** and **condition1**.
What will this program do?
## Example 1

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>6</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>12</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12</td>
<td>18</td>
<td>24</td>
<td>30</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>14</td>
<td>21</td>
<td>28</td>
<td>35</td>
<td>42</td>
<td>49</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td>24</td>
<td>32</td>
<td>40</td>
<td>48</td>
<td>56</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>18</td>
<td>27</td>
<td>36</td>
<td>45</td>
<td>54</td>
<td>63</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
</tr>
</tbody>
</table>

Output
public class example1 {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
            for (int j = 1; j <= 10; j++) {
                System.out.printf("%3d ", i*j);
            }
            System.out.printf("\n");
            if (i == 5) {
                break;
            }
        }
    }
}
### Example 2

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>16</td>
<td>18</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>9</td>
<td>12</td>
<td>15</td>
<td>18</td>
<td>21</td>
<td>24</td>
<td>27</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td>24</td>
<td>28</td>
<td>32</td>
<td>36</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td>30</td>
<td>35</td>
<td>40</td>
<td>45</td>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>
The Circle Program, Revisited.

- It would be nice if the user could input multiple values (and see multiple results) without having to rerun the program.
- This is the previous version. How can we change it?

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        System.out.printf("Please enter the radius: ");
        double radius = in.nextDouble();
        double circumference = 2 * Math.PI * radius;
        double area = Math.PI * Math.pow(radius, 2);
        System.out.printf("The circumference is %.2f.\n", circumference);
        System.out.printf("The area is %.2f.\n", area);
    }
}
```
The Circle Program, Revisited.

• First take: an infinite loop.
• Any room for improvement?

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        while (true) {
            System.out.printf("Enter the circle radius: ");
            double radius = in.nextDouble();

            double circumference = 2 * Math.PI * radius;
            double area = Math.PI * Math.pow(radius, 2);
            System.out.printf("Circumference = %.2f.\n", circumference);
            System.out.printf("Area = %.2f.\n\n", area);
        }
    }
}
```
The Circle Program, Revisited.

- First take: an infinite loop.
- Any room for improvement? User has no way to quit.

```java
import java.util.Scanner;

public class example1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        while (true) {
            System.out.printf("Enter the circle radius, or -1 to quit: ");
            double radius = in.nextDouble();
            double circumference = 2 * Math.PI * radius;
            double area = Math.PI * Math.pow(radius, 2);
            System.out.printf("Circumference = %.2f.\n", circumference);
            System.out.printf("Area = %.2f.\n\n", area);
        }
    }
}
```
• Second take: an infinite loop, with quit option.
• Any room for improvement?
• Second take: an infinite loop, with quit option.
• Any room for improvement?

<table>
<thead>
<tr>
<th>Enter the circle radius, or -1 to quit: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circumference = 6.28.</td>
</tr>
<tr>
<td>Area = 3.14.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Enter the circle radius, or -1 to quit: 2.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circumference = 14.45.</td>
</tr>
<tr>
<td>Area = 16.62.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Enter the circle radius, or -1 to quit: -1</th>
</tr>
</thead>
</table>

Exiting...
• Second take: an infinite loop, with quit option.
• Any room for improvement?

Enter the circle radius, or -1 to quit: 5,2
Exception in thread "main"
   java.util.InputMismatchException
   at java.util.Scanner.throwFor(Scanner.java:864)
   at java.util.Scanner.next(Scanner.java:1485)
   at java.util.Scanner.nextDouble(Scanner.java:2413)
   at example1.main(example1.java:9)
Java Result: 1

Example Output 2
• Second take: an infinite loop, with quit option.
• Any room for improvement?
• Would be nice to not crash when the input is not valid.
• In general: programs need input validation.
  – That will be our next topic in this course.

```java
Enter the circle radius, or -1 to quit: 5,2
Exception in thread "main"
  java.util.InputMismatchException
  at java.util.Scanner.throwFor(Scanner.java:864)
  at java.util.Scanner.next(Scanner.java:1485)
  at java.util.Scanner.nextDouble(Scanner.java:2413)
  at example1.main(example1.java:9)
Java Result: 1
```

Example Output 2
Detour: Random Numbers

• To generate a random number:
  – At the beginning of your java code, you should use this import statement:

    import java.util.*;

  – Once in your program, you should do:
    Random rand = new Random();

  – Then, to get a random integer from 0 up to (and including) MAX, you should call:

    int random_pick = rand.nextInt(MAX+1);
Guessing a Number

• Write a program that:
  – Picks a random number from 0 up to and including 100.
  – Gets in a loop where:
    • The user is asked to guess the number.
    • If the user guesses correctly, the program terminates.
    • If not, the system tells the user if the correct answer is higher or lower than the guess.
import java.util.*;
public class guessing_game {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        Random rand = new Random();
        int pick = rand.nextInt(101); // Between 0 and 100.
        int attempt = 1;

        while (true) {
            System.out.printf("Try %d: Guess the number: ", attempt);
            int guess = in.nextInt();
            if (guess == pick) {
                System.out.printf("Correct!!!\n");
                break;
            } else if (guess < pick) {
                System.out.printf("Go higher.\n");
            } else {
                System.out.printf("Go lower.\n");
            }
            attempt++;     
        }
    }
}
Example Programs

• Summing integers from 1 to N, and variations.
  – Summing squares.
  – Summing multiples of 7.
  – Summing primes.

• Printing divisors of a number.

• Removing spaces, dashes, parentheses from a phone number (or a credit card number).

• Printing a pyramid using the * character.